

Patent Application
Docket Number: EMC-03-100-CIP2
Applicant: Haase et al.
EMC CONFIDENTIAL
Express Mailing Label No.EK900600327US

EMC-03-100CIP2

PATENT

APPLICATION FOR UNITED STATES PATENT

Title: System and Method for Managing Data Associated with Copying and Recovery Procedures in a Data Storage Environment

By: David Haase, Dennis Duprey, Michael D. Haynes, and Somnath Gulve

A portion of the disclosure of this patent document contains command formats and other computer language listings, all of which are subject to copyright protection. The copyright owner, EMC Corporation, has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and
5 Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

Field of the Invention

10 This application generally relates to data storage management, and more particularly to management related to copying or replication of data in a data storage environment

Related Application

15 This Application is a continuation in part of U.S. Patent Application Serial No. To Be Determined (Attorney Docket No. EMC-03-100) filed on September 29, 2003 and entitled; "System and Method for Managing Data associated with Copying and Recovery Procedures in a Data Storage Environment." This Application is also related to Serial
20 No. To Be Determined filed on even date with the present U.S. Patent Application and also entitled; "System and Method for Managing Data associated with Copying and

Recovery Procedures in a Data Storage Environment” (Attorney Docket No. EMC-03-100-CIP1).

Background

5

Computer systems may include different resources used by one or more host processors. Resources and host processors in a computer system may be interconnected by one or more communication connections. These resources may include, for example, data storage systems, such as the Symmetrix™ or CLARiiON™ family of data storage systems manufactured by EMC Corporation. These data storage systems may be coupled to one or more host processors and provide storage services to each host processor. An example data storage system may include one or more data storage devices, such as those of the Symmetrix™ family, that are connected together and may be used to provide common data storage for one or more host processors in a computer system.

15

A host processor may perform a variety of data processing tasks and operations using the data storage system. For example, a host processor may perform basic system I/O operations in connection with data requests such as data read and write operations. Host processor systems may store and retrieve data using a storage device containing a plurality of host interface units, disk drives, and disk interface units. Such storage devices are provided, for example, by EMC Corporation of Hopkinton, Mass. and disclosed in U.S. Patent No. 5,206,939 to Yanai et al., 5,778,394 to Galtzur et al., U.S. Patent No. 5,845,147 to Vishlitzky et al., and U.S. Patent No. 5,857,208 to Ofek. The host systems

20

access the storage device through a plurality of channels provided therewith. Host systems provide data and access control information through the channels to the storage device and storage device provides data to the host systems also through the channels. The host systems do not address the disk drives of the storage device directly, but rather, 5 access what appears to the host systems as a plurality of logical disk units. The logical disk units may or may not correspond to the actual disk drives. Allowing multiple host systems to access the single storage device unit allows the host systems to share data stored therein.

It is known that is desirable to copy or replicate data, including for a backup 10 operation. Once data is replicated, copied, or otherwise backed up it may be used for a recovery or restoration process, wherein the copy of the data is restored back onto the source. This may be done for a variety of reasons including failures. So for example, data may be restored from the last good copy. However, the inventors of the present invention discussed below have critically recognized that it would be advantageous to 15 protect such a last good copy itself from being corrupted during a copying, recovery or restoration process.

Summary

To overcome the problems of the prior art mentioned above and to provide
5 advantages also described above, this invention is a system and method for protecting
data during a copying, recovery or restoration process.

In one embodiment, the invention is implemented as steps in a process. In other
embodiments, implementations include a system, a computer program product, or an
apparatus, wherein each embodiment is configured for carrying out the steps involved in
10 the methodology.

Brief Description of the Drawing

The above and further advantages of the present invention may be better understood by referring to the following description taken into conjunction with the accompanying drawings in which:

Fig. 1 shows a block diagram schematic of an embodiment of a data storage environment including at least one data storage system and including program logic for carrying out the method embodiments of the present invention;

Fig. 2 shows another schematic block diagram including modules of the program logic;

Fig. 3 shows a computer-readable medium including computer-executable code including the program logic for carrying out method steps of the method embodiments of the present invention;

Fig. 4 shows another flow logic diagram of steps for carrying out the method embodiments of the present invention;

Fig. 5 shows an exemplary embodiment including the data storage system and program logic of Fig. 1;

Fig. 6 shows an example of what may be experienced during carrying out the methodology of this invention;

Fig. 7 shows another exemplary embodiment including the data storage system and program logic of Fig. 1 and illustrates an example of states that may be experienced during carrying out the methodology of this invention;

Fig. 8 shows an example of what may be experienced during carrying out the methodology of this invention;

Fig. 9 shows another exemplary embodiment including the data storage system
5 and program logic of Fig. 1 and illustrates an example of states that may be experienced during carrying out the methodology of this invention;

Fig. 10 shows another embodiment including the data storage system and program logic of Fig. 1 useful for handling potential data corruption; and

Fig. 11 shows a method involving the embodiment of Fig. 10 and including the
10 data storage system and program logic of Fig. 1 useful for handling potential data corruption.

Detailed Description of the Preferred Embodiments

Introduction

In the preferred embodiment, the embodiment operates in cooperation and may be
5 a part of computer software, operating the preferred EMC CLARiiON or Symmetrix
storage systems available from EMC Corporation of Hopkinton, Massachusetts, although
one skilled in the art will recognize that the invention may be used with other data storage
systems. In the preferred embodiment, EMC CLARiiON storage system implements
aspects of the present invention as part of software that operates with such a storage
10 system. In this preferred embodiment, a locally accessible mirror copy of a specified
Logical Unit (LU), or logical mirror copy is available for backups or other host processes.
The logical mirror copy may also be useful for operations that change data, such as
production testing or data warehousing, without disrupting the production database.

Terminology

Some terminology used herein is shown in here in Table 1:

<i>Source</i> : Typically a logical unit (LU) from where data is mirrored or copied, and which herein is sometimes referred to as being cloned.
<i>Clone</i> : Another LU, other than the Source LU, but which is a copy of the specified Source.
<i>Clone Group</i> : A collection of a Source and all of its Clones.
<i>Sync Engine</i> : A part of the Program Logic that copies data between a Source and a Clone
<i>Synchronization</i> : Process of updating the data contents of Clone with that of its Source, preferably performed by the Sync Engine
<i>Reverse Synchronization (Reverse Sync)</i> Process of updating the data contents of a Source from one of its Clones, preferably is performed by the Sync Engine
<i>Extent</i> : An extent is a number of contiguous data blocks on an LU that is represented by a specific amount of data, e.g. a bit in a map, e.g. a bitmap
<i>Copy On Demand (COD)</i> : Process of copying a specific extent from the Clone to the Source, as occasionally required during a Protected Restore
<i>COD Engine</i> : A part of the Program Logic that coordinates and performs all COD.
<i>Fracture</i> : Process of logically disconnecting or discontinuing mirroring of data from the Source to a Clone (also known as splitting-off a Clone).
<i>Delta Map</i> : A map (e.g. a bitmap) that represents the extents that are different between the Clone and its Source.
<i>COD Queue</i> : This is a component of the Protected Restore that coordinates all the I/O requests that require data copied from the Clone LU to the Source LU before it can be processed.
<i>COD Scoreboard</i> : This is a component of the Protected Restore that allows the COD Engine and Sync Engine to coordinate which segments of the device are currently being copied either by the reverse-sync operation or by a COD operation.
<i>Recovery Process</i> : These are the steps that are performed to resume a Protected Restore that stopped because of a system failure (e.g., a re-boot). These steps are typically performed when the system boots up
<i>Persistent Delta Map</i> : The Delta Map stored on disk that is used for all the Clones in a Clone Group. It is used during a Recovery Process to fill the Delta Maps for all the Clones in a Clone Group.
<i>Protected Restore Map</i> : A map (e.g., a bitmap) used to track the extents of the Source LU that have been modified during a Protected Restore. This map is kept in memory as well as persistently for recovering from failures.
<i>Clone's I/O Manager</i> : This is the component that processes host I/O requests received by both the Source and Clone

1:

In a preferred embodiment, an important feature is a process denominated as a “Protected Restore”. This process involves a reverse-synchronization by copying the data content on the mirror copy back to the Source. But, due to the critical recognition of the inventors, unlike a prior art reverse-synchronization, the content of the mirror copy is protected during a Protected Restore. The ability to perform a Protected Restore is advantageous to a user. The user may logically disconnect or fracture the mirror copy from its Source, it represents a full point-in-time copy of the user’s production data. This fractured Clone can then be used to restore the Source with the data from a previous point-in-time (prior to the corruption of the production data). By preserving the content of the Clone during the Protected Restore, future restores using the same Clone result in the same point-in-time copy of data being copied to the Source.

Because of the various actions involved and taken care of during a Protected Restore, various internal mechanisms are included in embodiments of the invention to enable their implementation. These include a (1) Protected Restore map useful for (a) tracking changes during a Protected Restore; and (b) resuming an interrupted Protected Restore; and (2) a Copy-On-Demand (COD) Mechanism useful for processing Host I/O requests in a timely manner.

The preferred EMC embodiment may operate with EMC's TimeFinder and SnapView local replication applications, which eliminate the impact to production activities by using Business Continuance Volumes (BCV's) (discussed in the
5 incorporated '497 patent referenced below) or Snapshots as source volumes so applications stay online throughout the data movement process. However, the present invention may be used without requirement of such BCV's or Snapshots. For the sake of completeness, operational features embodied in EMC's Timefinder and Symmetrix are described in U.S. Patent 6,101,497 issued Aug. 8, 2000, and also in U.S. Patent 5,206,939
10 issued April 27, 1993, each of which is assigned to EMC the assignee of this invention and each of which is hereby incorporated by reference. Consequently, the following discussion makes only general references to the operation of such systems.

Preferred Embodiment Description

15

Referring to Fig. 1, Data Storage Environment 10 includes a Production Host 14 with an Interface 17, e.g. a graphical user interface a.k.a. GUI or command line interface a.k.a. CLI, or any type of communication link including a network link for communicating with Data Storage System 16 across path 15, and in particular for
20 accessing Production Data 20. Interface 17 may enable communication with the Data Storage system based on user or computer-generated commands.

Reference is made now to Figs. 1 and 3. Generally, in a preferred embodiment the maps in memory are used by the Program Logic 34 in cooperation with the operating system 32, and the CPU 31 on the source data storage system 16. The maps and Program logic operate in electronic memory 37 and when executed by CPU 31 over

5 communication path 39 carry out method steps embodying the invention. It is preferred that the Program Logic be computer software although it is possible for it to be embodied in whole or part in hardware or firmware. Program Logic 34 may also be embodied on a computer-readable medium 150 as shown in Fig. 3 and wherein the Logic is encoded in computer-executable code configured for carrying out steps of a method embodiment of

10 this invention. The methods and apparatus of this invention may take the form, at least partially, of program code (i.e., instructions) embodied in tangible media, such as floppy diskettes, CD-ROMs, hard drives, random access or read only-memory, or any other machine-readable storage medium, such as Medium 150 (Fig. 3) encoded with Program Logic 34. When the program code is loaded into and executed by a machine, such as a

15 computer, the machine becomes an apparatus for practicing the invention. The methods and apparatus of the present invention may also be embodied in the form of program code that is transmitted over some transmission medium, such as over electrical wiring or cabling, through fiber optics, or via any other form of transmission. It may be implemented such that herein, when the program code is received and loaded into and

20 executed by a machine, such as a computer, the machine becomes an apparatus for practicing the invention. When implemented on one or more general-purpose processors,

the program code combines with such a processor to provide a unique apparatus that operates analogously to specific logic circuits.

Referring to Figs. 1 and 2, a Data Storage Environment 10 includes a Production Host 14 with an interface 17 that may be for example a graphical user interface (GUI) or command line interface (CLI). A user, or some form of automated process may invoke processes of the present invention through interface 17 and communication path 15, and such processes may include the Protected Restore feature discussed above. In general, in a preferred embodiment, when a user initiates a Protected Restore, discussed further with reference to Fig. 4 below, the Program Logic 34 which includes the Sync Engine 41 (Fig. 2) begins a reverse-sync along path 21 to copy data from the Data Copy or Clone 22 to the Production Data or Source 20 in memory 33. Although the Clone is shown on the same system as the Production Data, it may be located on another Data Storage System. Program Logic 34 also includes Clone I/O Manager 45 and Collision Detector 47, which are discussed below with reference to Figs. 10 and 11.

Referring to Fig. 4, an overview of a methodology embodying at least some aspects of the invention is shown. In step 40, the Protected Restore Process is initiated. A reverse-sync procedure automatically begins in step 42. Different extents required for such a procedure are copied in step 44. A presentation of the data on the source is presented in an apparent instant fashion in step 46, and the engines (Sync and COD) coordinate in step 48. The data is Protected in step 50 despite the risks involved during the copying. Each of these steps are discussed in more detail herein.

The Sync Engine uses the Delta Map 28 stored in memory 36 to preferably copy only the extents that are different between the Clone and its Source, and a Protected Restore Map 32 and also a Tracking Map 35 are used as described herein. The Sync Engine copies the required data, preferably, starting at the beginning of the Clone and working its way to the end of the LU. However, once the Protected Restore process has been initiated, the user may expect the data from the Clone to “instantly” appear on the Source (even if the Sync Engine has not copied the data yet). This “instant appearance” of data is accomplished via the Copy on Demand or COD Engine 43 of the Program Logic (Fig. 2).

When a host I/O request is received by the Source, the request will be placed in the COD Queue and the COD Engine will copy any required data areas (represented by bits set in the Delta Map 28) from the Clone to the Source in order to allow the host I/O request to be processed. Without the COD Engine copying this data on-demand, host I/O requests could time-out waiting for the Sync Engine to copy it. During the Protected Restore, the Sync Engine and COD Engine work together to coordinate the copying of data from the Clone to the Source. The Sync Engine works sequentially across the Clone LU copying the required data, while the COD Engine copies the required data to ensure the quickest processing of host I/O requests. These two engines use the Tracking Map denominated as a COD Scoreboard 35 to communicate with each other.

Unlike a regular Reverse-Sync, a Protected Restore ensures that the data on the Clone is not modified during the operation. During a regular Reverse-Sync, host write requests received by the Source are simply mirrored to the Clone while host read requests

are redirected and processed by the Clone. Both the Protected Restore Map 32 as well as the Delta Map 28 are used during the Protected Restore to protect the Source and Clone from data corruption while presenting an “instant” copy of the Clone’s data on the Source. It also ensures the tracking of those data areas that are different between the
5 Source and Clone for subsequent operations.

During a Protected Restore, the Sync Engine copies data from the Clone to its Source. During this reverse-sync, the Sync Engine uses the Clone’s Delta Map to determine which extents are required to be copied. If a bit is set in the Delta Map for a particular region, it means that the data in that region is different between the Clone and
10 its Source and must be copied. As extents are copied from the Clone to its Source, their corresponding bits are cleared from the Delta Map.

Because host I/O can continue to the Source during the Protected Restore, data may be written to it during the operation. These changes to the Source must be tracked for subsequent Synchronization or Protected Restore operations. The Protected Restore
15 Map is used to track those extents that are modified due to host write requests during the Protected Restore. Once the Sync Engine completes the reverse-sync, this Protected Restore Map becomes the new Delta Map for the Clone.

When a host-write request is received for the Source LU during a Protected Restore, a determination is made whether a COD is required prior to processing the
20 request. This determination is made by checking both the Delta Map and the Protected Restore Map. If there are bits set in any of the extents of the Delta Map that represent those regions affected by the host I/O request (read or write) and the corresponding bit is

NOT set in the Protected Restore Map, then a COD is required. After each required COD has completed, the bit that corresponds to the extent affected by the COD is cleared from the Clone's Delta Map. This is done to ensure that the Sync Engine does not subsequently copy the data from the Clone to the Source which would cause a host write request (which triggered a COD) to later be overwritten with data from the Clone.

After each COD required for a host write request has been completed, the bit(s) that correspond to the extent affected by the request are set in the Protected Restore Map. This is done to track the changes between the Source and Clone.

Referring to Fig. 5, an exemplary embodiment is shown to further describe the operation of the invention. Production Host 14 includes interfaces 17a-17c involved with Host-write 1-3, respectively over communication path 15a-c, respectively to source 20, and wherein Reverse-Sync's are enabled over path 21 for Clone 22, and path 24 may also be used for operations.

Referring now to Fig. 6, the respective maps 28-35 are shown as they change after the respective Host-write requests over interfaces 17a-c. Write 1 does not require a COD because bit (0) is not set in the Clone's Delta Map 28. Write 1 is processed and bit 0 is set in the Protected Restore Map 32 to indicate that the data has been modified in the extent represented by bit 0. Write 2 and 3 both require CODs because bits 4 and 7 are set in the Clone's Delta Map but not in the Protected Restore Map. Both Write 2 and 3 are put on the COD Queue 51 while a COD is performed on the extents represented by bits 4 and 7 in the Delta Map. After the CODs, bits 4 and 7 are cleared from the Clone's Delta

Map. Bits 4 and 7 are then set in the Protected Restore Map and Write 2 and 3 are removed from the COD Queue and are processed.

Referring to Fig. 7, an example of embodiments of the invention operating in the case of a failure scenario is described. Delta Maps and the Protected Restore Map are used to protect the Source 20 from data corruption. The following example illustrates the state of a Clone Group (Source 20 and Clones 22 and 60) during a Protected Restore. In this example, the Sync Engine of the Program Logic copies the required data (using the Delta Map 28) from Data Copy or Clone 22 to the Source 20 while Data Copy or Clone 60 is in a fractured state (represented by broken line designation for path 58 from the Source to Clone 60). For the sake of completeness, Reverse-Sync operations from Clone 22 to Source 20 occur over path 21, and COD's occur over path 23 or 25, as described for a non-failure type scenario. However since a new Clone 60 is being employed, a Delta Map 81 for that Clone and a Persistent Delta Map 82 which indicates the state at finish is also shown.

During this operation, Host-write 1 though interface 17a is received by the Source 20 over path 15a. This write request affects a data area on the Source LU that is represented by multiple (2) bits in the Clone's Delta Map (bits 6 and 7). Because bit 6 of Clone1's Delta Map 28 is not set, no COD is required. However, the Host write request still cannot be processed at this time until bit 7 is checked. Because bit 7 of Clone1's Delta Map 28 is set and the corresponding bit is not set in the Protected Restore Map 32, a COD is required. The COD Engine copies extent 7 from the Clone 22 to the Source 20.

Referring to Fig. 8, once the COD has been completed, Host-write 1 can be processed. Before doing so, bits must be updated in the Protected Restore Map as well as Clone 60's Delta Map 81. The bits (6 & 7) in the Protected Restore Map 32 that correspond to those extents on the Source LU affected by the Host-write 1 must be set to
5 identify the data areas to protect from data copies performed by the Sync Engine that might otherwise corrupt data by overwriting. The bits (6 & 7) in Delta Map 81 are set to track those extents with data that is different between Clone 60 and the Source 20. The data from the Delta Maps 28 & 81 are written to persistent storage and the write request is processed. The Persistent Delta Map 82 now contains a bit set for those extents with
10 data that is different between the Source 20 and each of its Clones 22 & 60.

Referring now to Fig. 9, prior to a Protected Restore being resumed during the Recovery Process, the Protected Restore Map 32 is filled using a Persistent Protected Restore Map 83 kept in persistent storage (e.g., on disk). Both Clone 22's and Clone 60's Delta Maps are filled using the Persistent Delta Map 82. Because there is a single
15 Persistent Delta Map used to represent multiple Clones, it is possible that the Restore Process will cause a bit to be set in Clone 22's Delta Map that had previously been cleared either by the Sync Engine or the COD Engine. In the exemplary embodiment of Fig. 9, it can be seen in area 63 that bits 6 and 7 in Clone 22's Delta Map 28 have been set during the Recovery Process because those bits had been set in Clone 60's Delta Map
20 81 and therefore were set in the Persistent Delta Map 82. These bits were not set prior to the failure which lead to the recovery.

So, when the Protected Restore is resumed, the Sync Engine begins copying the required extents (using the Delta Map 28) from Clone 22 to the Source 20. If the Sync Engine copied the extents represented by bits 6 and 7, then effect of the Host-write request would get overwritten, which would be undesirable data corruption. But
5 reference to the Protected Restore Map 32 confirms that the corresponding bits are set in the Protected Restore Map, so the corresponding data areas should be protected on the Source. Accordingly, the Program Logic will control the Sync Engine and the COD Engine so neither will cause copying of these data areas from the Clone to the Source.

Also, If Host Write 2 (shown at 17a of Fig. 9) is received after the Protected
10 Restore has been resumed, a COD should not be performed even though the bits that represent the affected extents (bits 6 and 7) are set in the Clone's Delta Map 28. Doing so would also cause Host Write 1 to get overwritten, which would corrupt the data on the Source..

To provide the "instant" appearance of Clone data on the Source (without data
15 corruption), the Clone's Delta Map and the Protected Restore Map are used. When a read request is received by the Source during a Protected Restore, a check is made to determine if the bits are set in the Clone's Delta Map that correspond to extents affected by the read request. If the bit is set and the corresponding bit is not set in the Protected Restore Map, a COD is performed to copy that extent from the Clone to the Source. The
20 bit is then set in Protected Restore Map and cleared from the Clone's Delta Map. Once all the required data has been copied from the Clone to the Source, the host read request can be processed.

As mentioned earlier, some host I/O requests require a COD before the request can be processed. Waiting for the Sync Engine to copy the required extent may take quite a while because of the serialized nature of the way it copies data. This could result in host I/O requests timing out.

5 When a host I/O request requires data to be copied from the Clone to the Source prior to it being processed, the request is placed on the COD Queue. A check is then performed to determine if there are any extents that need to be copied that are not already in progress by the Sync Engine or COD Engine. This is done by checking the COD Scoreboard. Prior to copying an extent from the Clone to the Source, the Sync Engine
10 and COD Engine set a bit in the COD Scoreboard for the extent they are copying. This allows both engines to know which extents the other (and themselves) is copying.

 If there are no required extents to be copied that are not already in progress, the I/O request remains in a COD Queue until the required extents (that are already in progressed) are copied. If there are extents that need to be copied that are not already in
15 progress by either engine, a COD is performed and the corresponding bit is set in the COD Scoreboard.

 After each extent is copied from the Clone to the Source (either by the Sync Engine or COD Engine), the corresponding bit is cleared from the respective Clone's Delta Map and from the COD Scoreboard. A check is then made to see if any I/O
20 requests on the COD Queue can be processed. If there are no bits set in the Clone's Delta Map for those extents affected by the I/O request, the request can be processed. If there are bits set, more extents need to be copied before the request can be processed. A check

is again made to see if any required extents are already in progress. If all required extents are already in progress, the request is left on the COD Queue. If an extent needs to be copied that is not already in progress (corresponding bit not set in the COD Scoreboard), a COD is performed for the extent. Processing of the COD Queue in this manner allows
5 host I/O requests to be processed as soon as their required extents have been copied from the Clone to the Source.

Another embodiment of the invention is shown in Figs. 10 and 11. During a Reverse-Synchronization, host write requests to the Source are being “mirrored” to the Clone to ensure the data contents of the Clone is the same as its Source. The inventors
10 have critically recognized that it is possible that the host write data being “mirrored” to the Clone may be involved in a collision with the data region being copied from the Clone to the Source as part of the Reverse-Synchronization. This collision could result in a data corruption on the Source. An embodiment including a process for handling such a collision and preventing data corruption is discussed below.

15 Reference is made now Figs. 10 and 11, wherein an exemplary embodiment and method steps are shown including elements of Fig. 1. In this example, when a user initiates a Reverse-Synchronization over path 21 through for example an interface 17d from host 14, the Sync Engine of the Program Logic begins to copy data over path 21 from the Clone 22 to the Source 20. Preferably, as a performance optimization, the Sync
20 Engine uses the Clone’s Delta Map 97 (Fig. 11) to copy only the extents that are different between the Clone and its Source. The Sync Engine sequentially copies the required data starting at the beginning of the Clone and working its way to the end of the LU.

Once a Reverse-Synchronization operation has been initiated, it is reasonable to expect host write 1 request over 17d sent to the Source 20 over 15a to be mirrored to the Clone 22. This is done to ensure the data contents on the Clone are identical to that of its Source. But the inventors have recognized that without embodiments of this invention, it could be possible that host write data is “mirrored” to the Clone while the same data region is copied from the Clone to the Source as part of the Reverse-Synchronization shown in Fig. 10.

Without the present invention, it would also be possible that host write data would be written to the Source while the same data region is being written to the Source with data it read from the Clone as part of the Reverse-Synchronization. This “collision” of host write data with that of the data being Reverse-Synchronization could result in a data corruption on the Source, but this invention solves this problem in the embodiment discussed herein with reference to Figs. 10 and 11. A collision may occur when a host write request to the Source is being “mirrored” to the Clone to ensure the data contents of the Clone is the same as its Source. If the write data is being mirrored to a data region on the Clone at the same time the Sync Engine is reading this same data region as part of a Reverse-Synchronization operation, the data read from the Clone may contain incomplete “mirrored data” along with old data which will then be written to the Source. This would overwrite the good data already written to the Source by the host write request causing a data corruption on the Source LU.

Reference is made below to Fig. 11. To handle these collision scenarios, the Program Logic is enabled to detect a collision. The Clone’s I/O Manager 45 and Sync

Engine 41 call the Collision Detector 47 to perform this detection (see Fig. 2) While processing a Host Write Request for Data Region # 3 in step 90 received by the Source during a Reverse-Synchronization, the Clone's I/O Manager passes the write request to the Collision Detector in steps 92 and 94. The Collision Detector in step 94 determines
5 the extents on the Source that are affected by the host write request. If any of the affected extents are the same as the extent currently being copied from the Clone to the Source as part of the Reverse-Synchronization, the Collision Detector of the Program Logic instructs the Sync Engine in Step 98 to set a property that indicates that the extent needs to re-copied. The property may be any indicator, such as a local variable in memory 37
10 and setting such variables is known. Since the Sync Engine only copies one extent at a time, it is not important to indicate which extent collided, just that a collision occurred.

Similarly, while a Reverse-Synchronization is in progress, the Sync Engine passes the extent that is to be copied to the Collision Detector. The Collision Detector determines if there are any host write requests along path 15b that are currently being
15 processed by the Clone's I/O Manager 45 that affect the specified extent. If there are, the Collision Detector tells the Sync Engine to set a property that indicates that that extent needs to be re-copied along path 99. Path 21 and 99 may be coincident and are shown separately to point out the operations taking place.

At the completion of each extent from the Clone 22 to the Source 20 as part of the
20 Reverse-Synchronization along path 21, the Sync Engine of the Program Logic checks if the extent needs to be re-copied. If it does, the Sync Engine will re-copy the extent from

the Clone to the Source. By handling these collision scenarios with a retry method (see e.g., step 96), the data on the Source is not corrupted.

This method also allows host write requests to be processed as they are received which results in efficient host write performance. Other methods that may occur to one skilled in the art may include queuing the host write request when a collision is detected until the data region has been copied from the Clone to the Source LU. However, such queueing of the write requests in this manner may affect performance.

The embodiments described above may be selectively employed separately or combined to function cohesively. For example, it may be advantageous to use a collision detection method with restore, or provide a selective adaptation wherein the collision detection method is used selectively, or a protect restore is used instead. It is advantageous to provide program logic that provides all of the above-described functionality so that selective options are available.

Having described a preferred embodiment of the present invention, it may occur to skilled artisans to incorporate these concepts into other embodiments. Nevertheless, this invention should not be limited to the disclosed embodiment, but rather only by the spirit and scope of the following claims and their equivalents.